A modeling language for 3D process plant layout representation, exchange and visualization

Thomas Paviot[1], Virginie Fortineau[1], Samir Lamouri[1] and Ludovic Louis-Sidney[2]

LCPI/Arts et Métiers ParisTech, Paris, France.
[2] LISMMA/Supméca, Saint-Ouen, France.

Abstract. In the nuclear industry, achieving Long Term Data Preservation is a requirement for nuclear power plants to be safely built, operated over five or six decades and retired. Among them, CAD data suffers from some strong dependencies on the software vendors and its data model thus leading to a possible weakness in the preservation workflow. This paper presents a modeling language, suitable for the 3D representation of a process plant layout, based upon a procedural Constructive Solide Geometry (CSG) approach. The language execution, as well as the layout rendering and exchange, are experimented using a platform independent implementation, based on free software and open standards.

Keywords: 3D Plant Layout, Computer Aided-Design, Constructive Solid Geometry, STEP, COLLADA, WebGl.

1 Industrial context

In the nuclear industry, designing power plants requires to ensure the long term data preservation (LTDP) of the 3D Computer Aided-Design (CAD) data that are generated by the engineering teams, due to strong safety regulations and because of the long lifecycle nature of this complex system (around one hundred years from early design, to building, operation and maintenance up to the final retirement). This research focuses on some specific stage of the early power plant engineering: the layout design. According to Theodosiou *et al.* [1], the layout of complex products and systems "describes the system's structure and components as well as related procedures and constraints". Facilities layout is concerned with the spatial arrangement of a set of departments or equipment items . This is an important stage at the design level, which often results in a complex problem due to the high number of decisions involved [2]. This layout design takes place at the conceptual design stage [3] and is used for early design reviews related to regulation checks, first design decisions or preliminary simulations. The main asumption of this paper is that the 3D representation of the layout only needs simple boundary volumes that constraint all the subcomponents that are further part of the plant.

As underlined by Lorie [4], the challenge of LTDP is "to ensure that the information, generated today, can survive long term changes in storage media,

devices and data formats". This paper focuses on the latter: how to keep control over the 3D data format representing the plant layout? In other words, the purpose is to get free from any CAD package vendor for this representation.

Open standards are known to be *a priori* suitable regarding this issue [5]. For instance, The *STandard for the Exchange of Product data* (STEP) [6] provides a *Boundary Representation* (BRep) description of a shape topology/geometry that can be transfered from one CAD package to another. However, this transfer is not conservative from a semantical viewpoint: a lot of information is lost during the conversion, especially the design intent or *parametric representation* of the shape. This semantic loss occurs whatever the standardized neutral format may be (IGES for surfacic representation, STL and VRML mesh based representation etc.), *i.e.* these standards only allows to exchange a snapshot of the 3D model without any information about the design process that led to this topological/geometrical state. These models are "difficult or impossible to edit" [7] which is in contradiction with Lorie's challenge. Other approaches then need to be investigated.

As a consequence, the paper is structured as follows: section 2 deals with the literature review related to macro parametric and procedural approaches in the CAD area. Section 3 proposes a modeling language suitable for the 3D layout design. The language, its execution and a visualization workflow are experimented in section 4. Finally, section 5 concludes the paper.

2 Macro-parametric design methodology

This section explores the macro-parametric approach as a potential solution to the issue presented above. According to Yang *et al.* [8], a macroparametric method describes the parametric information of a CAD model as "a design command sequence", here defined as a procedural approach. Past researches dealing with this topic can be classified into two categories:

1. Product data exchange or macroparametric approach as a way to exchange models between different CAD packages: for instance, Choi *et al.* [9] proposed a macroparametric approach to exchange CAD models in an heterogenous environment. This approach is further extended to a feature-based macrofile format supporting the representation of the history-based parametric design [10]. These works are based upon an XML files description. Pratt *et al.* [7] enriched the STEP standard semantics with parametric modeling features [11].
2. Synchronized collaborative design or macroparametric approach as a way to enable wide scale and real-time design collaboration: the exchange of a command sequence is here considered as an efficient way to reduce the necessary network bandwith compared with the one required to transfer netural format files. For instance, Li et al. [12] introduce a set of neutral CAD modeling command for real-time and wide collaboration.

Researchers who developed MP approaches used Parametric Feature Modeling (PFM) concepts and defined a set of commands that map a small common subset

from the most famous CAD packages available on the market. These works suffer from three major drawbacks:

- incompleteness of the set of commands: whether it is to exchange models or to enable wide-scale real-time collaboration in an heterogeneous environment, the common features between CAD packages are restricted to basic operations (for instance *Sketch*, *Protrusion*, *Revolution* etc.). Indeed beyond basic functions proposed by every CAD package, many advanced functions are specific to a software: the receiver system won't be able to rebuild the model since it is missing the function,
- the PFM approach results in a high number set of commands that rely on a complex data model: the XML or ASCII file size that describes a model instance might be huge,
- the XML description is not human readable, as well as a set of EXPRESS instances serialized to an plain-text ASCII or XML file. These files cannot be edited and modified by a designer who would like to modify the geometry of the model. The modification has to be done from a CAD software.

For these reasons, current MP approaches are not judged suitable for the present research: the plant layout 3D model should be described in a simple way, be human readable and modified from outside any CAD package. The next section proposes a solution to overcome these issues.

3 Constructive Solid Geometry modeling language for 3D preliminary design

This section introduces a procedural *Constructive Solide Geometry* (CSG) [13] modeling language. The idea of CSG is to combine simple 3D shapes to more complex ones with boolean operations in 3-dimensional space. CSG modeling has been superseded by PFM in most of the usual current CAD packages, however CSG is interesting in the current study case because only basic shapes are needed for the plant layout representation and because CSG is based upon a small set of shapes/transformations leading to a lightweight model specification. Subsection 3.1 introduces the language semantics and subsection 3.2 deals with its syntax.

3.1 Model semantics

The semantics of the language is based upon the geometry foundation, the geometry factory and the layout factory.

The Layout Factory: this is the top-level component, composed of the base `Layout` class, which is a container for the geometry, sublayouts, and their absolute placements in the 3D space.

The geometry foundation: composed of ten basic classes: `Point` (a cartesian point (x, y, z)), `Vector` (from its cartesian coordinates (v_x, v_y, v_z)), `Matrix` (a $4*4$ array), and six basic shapes (`Box`, `Cylinder`, `Sphere`, `Cone`, `Torus`, `Wedge`) that inherit from an abstract `Shape` class (see figure 1).

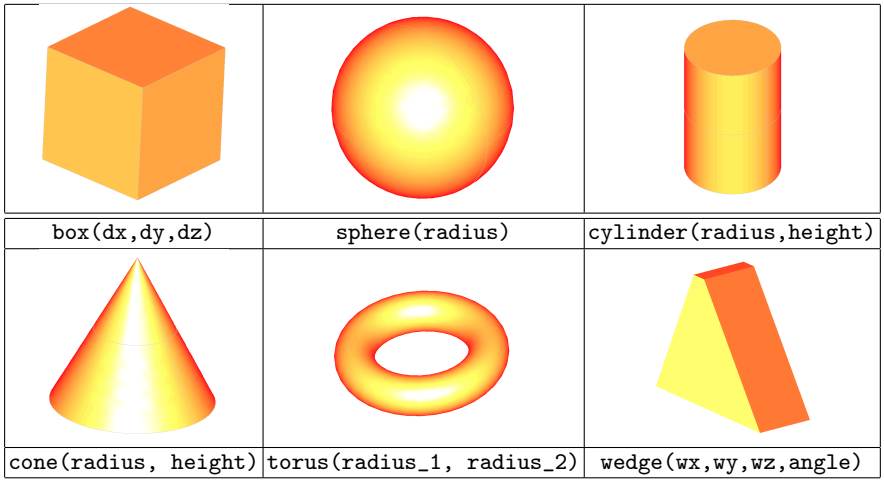


Fig. 1. Geometry factory: basic shapes instanciation

The geometry factory: a set of functions enabling a procedural treatment. These functions allow to instanciate classes, create geometry and perform boolean operations to generate the final geometry. The geometry factory provides three boolean operations that take two shapes as arguments and return the result shape: `fuse`, `cut`, `common`. Table 1 illustrates the boolean operations of two overlapping shapes A and B utilising Venn diagrams. A set of three transformations (`translate`, `rotate`, `scale`) allows to moving or deforming shapes. Table 2 illustrates these 3D shape transformations (they are presented on a 2D projection view on table 2).

3.2 Syntax definition

The purpose is to make the previous set of commands be computer-interpretable, the definition of an unamiguous syntax is thus mandatory. The syntax of a programming language is the set of rules that define the combinations of symbols that are considered to be correctly structured programs in that language [14] (variable affectation, conditionnal expressions etc). The resulting model has to be human readable, easily understandable and usable by a design engineer and

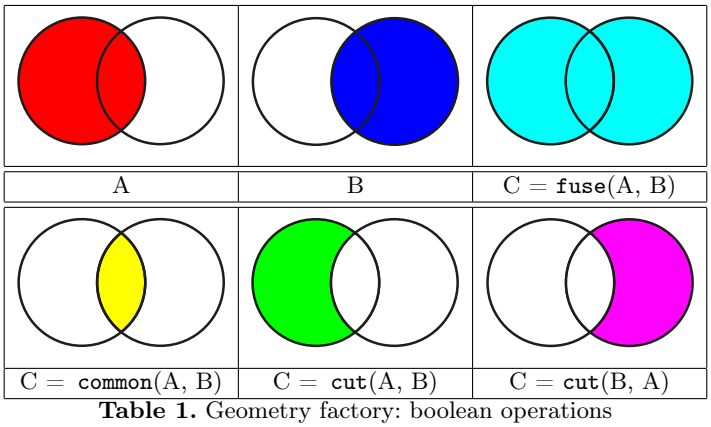


Table 1. Geometry factory: boolean operations

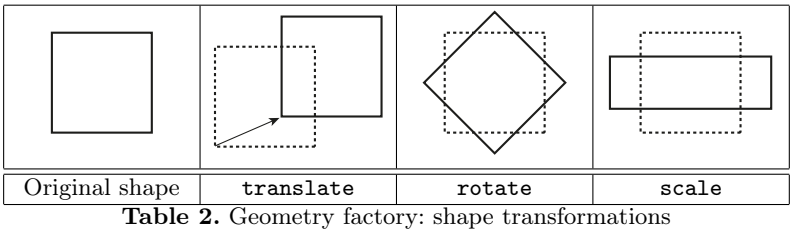


Table 2. Geometry factory: shape transformations

must produce the smallest program size as possible, in order to ensure that the 3D layout definition is actually robust over a long time period (it must be reusable for decades). The definition of a specific syntax requires the development of a robust lexer and parser: developping a new syntax from scratch or use an existing one has to be balanced regarding this effort and the previous criterions. High level programming languages, such as Java, Scheme or Python, are potential candidates: they provide a strong abstraction over machine language and a robust and mature platform-independant implementation. This is experimented in the next section.

4 Experiments

Experiments presented in this section deal with a basic example (4.2), measurement of the language intepreter performance (4.3) and an exchange/visualization workflow (4.4). Results are discussed in subsection 4.5.

4.1 Language syntax choice

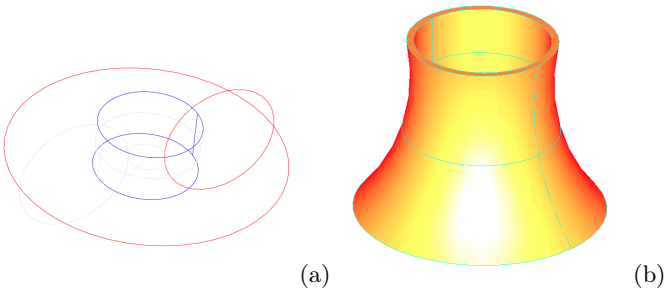
As discussed at the end of the previous section, the decision was took to use the syntax of some existing high-level interpreted language: the Python programming language [15], depicted by Dubois [16] as “clear and intuitive for engineers

and scientists". The language interpreter was implemented using only free and open source software libraries: pythonOCC[1] and OpenCascade™ Technology library[2]. Following tests were performed on a Mac OSX machine running 2 64bit cores.

4.2 Basic test for the CSG engine

The simple following model aims to represent a cooling tower, considered as the result of a set boolean operations involving a cylinder and a torus. The following program produces the output as presented on figure 2 (units are in meter). Figure 2(a) shows the basic shapes (cylinder,torus, presented in a wireframe mode) that are passed to the cut operation, the output shape is shown on figure 2(b).

```
cyl = cylinder(13,13)
tor = translate(torus(20,15),vector(0,0,10))
external_volume = cut(cyl,tor)
volume_to_remove = scale(external_volume,point(0,0,0),0.9)
cooling_tower_geometry = cut(external_volume,volume_to_remove)
```



(a) (b)

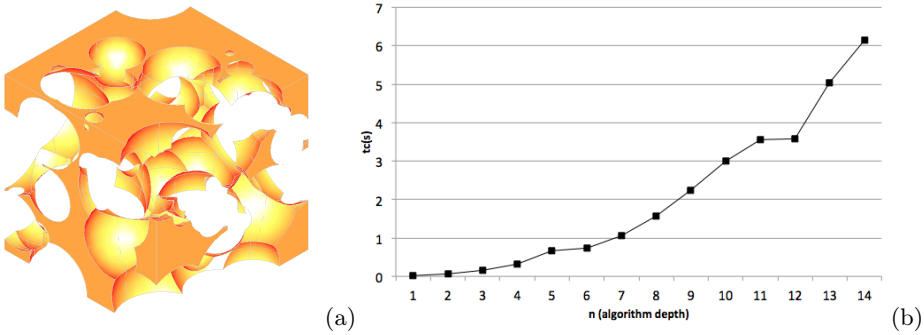
Fig. 2. Model for a cooling tower

4.3 Measure of the CSG interpreter performance

A plant layout may be composed of thousands of components, each of them being the result of some shape creation, transformation and boolean operations. As a consequence, a fast and robust CSG engine is required. Two different stress tests were conducted: the first one deals with the boolean operations performance in terms of computing time and topology consitency, the second one with the creation of thousands of elementary shapes in terms of computing time and memory consumption.

[1] http://www.pythonocc.org
[2] http://www.opencascade.org

Boolean operations stress test: a recursive boolean cut with a sphere was performed from an initial cube. The results are presented on figure 3: figure 3(a) is a screenshot of the final result after removing 20 random spheres. Since sphere radius and location are randomly setted, the resulting geometry looks different each time the test is processed: the topology consistency has been checked for all runs. Figure 3(b) presents a chart that reports the computing time t_c in seconds according to the number n of requested boolean operations, *i.e.* the algorithm depth. $t_c(n)$ appears to be an exponential function, which can be explained by the fact the topology complexity increases, by a factor f, each time a new cut operation is performed over the previous topological state.



(a) (b)

Fig. 3. CSG engine stress test for boolean operations

Multiple instanciation stress test: this test aims to checking the ability of the engine to create many instances of elementary shapes in a reasonable time with a reasonable memory consumption. "Reasonable" here means tha the test should be performed on a personal computer in less than a few minutes while consuming less than 1Gb of memory. This test case creates n random basic geometries (toruses, boxes, spheres) and measures the computing time as well as the memory consumption (see results in table 3). $t_c(n)$ and $M_c(n)$ appear to be both linear functions of n.

n	10	10^2	10^3	10^4	10^5
computing time t_c in ms	$2,2$	15	140	$1,47$	$15,8$
memory consumption M_C in Mb	$0,47$	$0,7$	7	$87,7$	903

Table 3. Multiple geometry instanciation in a 3D layout

4.4 Exchange and rendering of the 3D layout

Beyond checking the feasibility of the modeling language to deal with many complex shapes, we investigated as well the possibility to exchange and visualize the resulting layout, so that this modeling approach can be integrating into usual design and checking PLM workflows. In this test case, we consider a basic layout composed of one shop floor and a pressure vessel. The 3D model for the pressure vessel is obtained from elementary volumes (spheres, cylinders and boxes). Regarding the exchange with other CAD packages, STEP is the most famous neutral format, especially the 203 and 214 *Application Protocols*. Moreoever, it is natively supported by usual CAD softwares for a long time. Ding *et al.* [17] report many different file formats for the exchange of tesselations. Among them, JT, 3DXML or PDF3D are the most known, but COLLADA[3] was further investigated since it the interest of being a free and open standard. At last, a way of rendering this tesselation using open standards was also experimented. The recent WebGl[4] standard brings 3D to the internet together with COLLADA [18]: we experimented the visualization of the 3D layout within a webbrowser supporting this new feature. This leads to a plugin free online visualization, which is much practical for distant project reviews or digital mockup investigation and inspection on computers that does not have any CAD package installed. At last, it would allow to render the layout on lightweight mobile terminals (*e.g.* smartphones, tablets) thus enabling *in situ* project reviews. The experiment result is presented on figure 4: the model, expressed using the modeling language presented in section 3, is processed by the engine, which generates a STEP file, a COLLADA 1.4 file (imported into Google SketchUp to demonstrate the file consitency) as well as a WebGl JavaScript Object Network (JSON) object served to a web client (Firefox 11).

4.5 Results and discussions

The stress tests demonstrated that it is possible to easily create a 3D bounding volume and generate the geometry: topology consistency, computing time and memory consumption allow running this kind of language on a personal computer. The visualization workflow described in the last subsection shows it is possible to store a tesselation using the COLLADA data model: the mesh size however still has to be reduced in order to provide a ligthweight tesselation to the renderer. The WebGl based renderer allows to checking the current WebGl implementations of modern webbrowsers in this context: they appear to be increadibly stable, mature and efficient, according to the other tests that were ran. This global workflow although has to be experimented for huge layouts involving dozens of thousands of components. To conclude this discussion, the model and the software prototype presented above provide a way to: describe a 3D

[3] http://collada.org
[4] http://www.khronos.org/webgl/

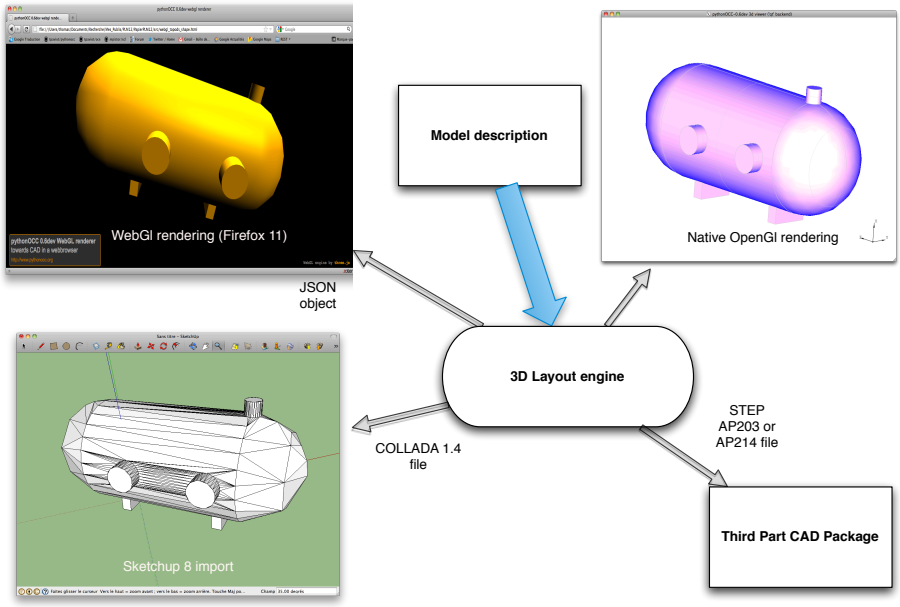


Fig. 4. Pressure vessel model ready for a PLM workflow

volume with a simple and human readable model, execute this model thanks to a language based upon a high-level programming language syntax, generates a platform independant visualization solution (WebGl/web browser) as well as a neutral representation (STEP) ready for the exchange with third part CAD tools and, at last, a COLLADA file ready for tesselation exchange. Although a CAD kernel is still required for supporting this workflow, the dependency over one specific CAD solution is removed: the control over all 3D layout assets is ensured.

5 Conclusion - Further works

This paper introduced a modeling language suitable for 3D process plant layout representation. This language uses a procedural Constructive Solid Geometry approach. Experimentations demonstrated that it can be executed in order to generate simple geometries intended to represent the boundary volume of complex components. Morover, the paper also contributes a workflow to enable exchange of the resulting geometry as a STEP file, visualization storage using the COLLADA standard as well as a webbrowser based visualization enabling online rendering. Further research purposes are to reduce the tesselation size, add parametric features in order to provide more flexibility and develop a business object ontology on top of this model.

References

[1] Theodosiou, G., Sapidis, N.S.: Information models of layout constraints for product life-cycle management: a solid-modelling approach. Computer-Aided Design **36**(6) (2004) 549–56

[2] Barbosa-Póvoa, A., Mateaus, R., Novais, A.Q.: Optimal 3D layout for industrial facilities. International Journal of Production Research **40**(7) (2002) 1669–1698

[3] Mantyla, M.: A modeling system for top-down design of assembled products. IBM Journal of Research and Development **34**(5) (1990) 636–659

[4] Lorie, R.A.: Long term preservation of digital information. Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries (2001) 346–352

[5] Rachuri, S., Subrahmanian, E., Bouras, A., Fenves, S., Foufou, S., Sriram, R.: Information sharing and exchange in the context of product lifecycle management: Role of standards. Computer-Aided Design **40**(7) (2008) 789-–800

[6] Pratt, M.J.: Introduction to ISO 10303: the STEP standard for product data exchange. Journal of Computing and Information Science in Engineering **1** (2001) 102–104

[3] Cellary, W., Walczak, K.: Interactive 3D Content Standards. In Interactive 3D Multimedia Content, ed. Cellary and Walczak, pub. Springer London, ISBN 978-1-4471-2497-9 (2012) 13–35

[7] Pratt, M.J., Anderson, B.D., Ranger, T.: Towards the standardized exchange of parameterized feature-based CAD models. Computer-Aided Design **37**(12) (2005) 1251–1265

[8] Yang, J., Han, S., Cho, J., Kim, B., Lee, H.: An XML-based macro data representation for a parametric CAD model exchange. Computer-Aided Design and Applications **1** (1-4) (2004) 153–162

[9] Choi, G.-H., Mun, D., Han, S.: Exchange of CAD Part Models Based on the Macro-Parametric Approach. International Journal of CAD/CAM **21** (2002) 13–-21

[10] Mun, D., Han, S., Kim, J., Oh, Y.: A Set of Standard Modeling Commands for the History-Based Parametric Approach. Computer Aided Design **35**(13) (2003) 1171–1179

[11] ISO 10303-111: Industrial automation systems and integration - Product data representation and exchange - Part 111: Integrated application resource: Elements for the procedural modelling of solid shapes, ICS: 25.040.40 (2007)

[12] Li, M., Gao, S., Wang: Real-time collaborative design with heterogeneous CAD systems based on neutral modeling commands. Journal of Computing and Information Science in Engineering **7** (2007) 113–126

[13] Lions, J.L., Pironneau, O.: Domain decomposition for CAD. Compte-rendu de l'académie des Sciences, Series 1, Mathematics **328**(1) (1999) 73–80

[14] Friedman, D.P., Wand, M.: Essentials of programming languages, 3rd edition. The MIT Press ISBN-13 978-0-262-06279-4 (2008)

[15] Lutz, M., Van, G.: Programming Python: Object-Oriented Scripting. pub. O'Reilly \& Associates, Inc. ISBN-10 0596000855 (2001)

[16] Dubois, P.F.: Ten good practices in scientific programming. Computing in Science & Engineering **1** (1999) 7–11

[17] Ding, L., Ball, A., Matthews, J., McMahon, C.A. and Patel, M.: Product representation in lightweight formats for product lifecycle management (PLM). Representations **September** (2007) 19–21

[18] Milivojević, M., Antolović, I., Ranĉić, D.: Evaluation and visualization of 3D models using COLLADA parser and WebGL technology. Proceedings of the 2011 international conference on Computers and computing (2011) 153–158